

IFW AF
2173

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:
Broussard

Serial No. 09/870,622

Filed: May 31, 2001

For: SYSTEM AND METHOD FOR
ENCAPSULATING SOFTWARE
COMPONENTS IN AN
APPLICATION PROGRAM INTERFACE
USING A PROXY OBJECT



Group Art Unit: 2173
Examiner: Bonshock, D.

Atty. Dkt. No. AUS920010260US1
(5468-07300)

I hereby certify that this correspondence is being deposited with the U.S. Postal Service with sufficient postage as First Class Mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313, on the date indicated below:

October 25, 2004
Date

Kevin L. Baffer

APPEAL BRIEF

Box AF
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313

Sir:

Further to the Notice of Appeal faxed to and received by the U.S. Patent and Trademark Office August 24, 2004, Appellant presents this Appeal Brief. The Notice of Appeal was filed following mailing of a final Office Action on May 27, 2004. Appellant hereby appeals to the Board of Patent Appeals and Interferences from a final rejection of claims 1-17 in the final Office Action, and respectfully requests that this appeal be considered by the Board.

I. REAL PARTY IN INTEREST

The subject application is owned by International Business Machines Corporation, a corporation having its principal place of business at New Orchard Road, Armonk, New York, 10504, as evidenced by the assignment recorded at Reel 011888, Frame 0535.

II. RELATED APPEALS AND INTERFERENCES

Notices of Appeal have been filed for the following applications, which share a common specification with the application currently on appeal.

09/870,615 Notice of Appeal filed 9/14/04

09/870,621 Notice of Appeal filed 9/24/04

However, because dissimilar art is cited in the present application and the above-mentioned related applications, Appellants do not believe that the outcome of this appeal will have any bearing on the Board's decision on the related appeals. No other appeals or interferences are known which would directly affect or be directly affected by or have a bearing on the Board's decision in this appeal.

III. STATUS OF CLAIMS

Claims 1-17 were originally filed in the present application. No amendments, cancellations, or additions to the claims were made in a Response filed February 23, 2004 to an Office Action mailed November 21, 2004. Claims 1-17 stand finally rejected under 35 U.S.C. § 103, which are the subject of this appeal. A copy of claims 1-17, as on appeal, is included in the Appendix hereto.

IV. STATUS OF AMENDMENTS

No amendments to the claims were filed subsequent to their final rejection. The Appendix hereto therefore reflects the current state of the claims.

V. SUMMARY OF THE INVENTION

Appellant's claimed invention relates to a system of software components (82, 84 and 88, Fig. 8), a computer-readable storage device (18, Fig. 1) and a method (Fig. 10) for graphically displaying an object (object 24, Fig. 2) created by an application program (APP 28) running under, and at least somewhat dependent on, an operating system (OS 40). In accordance with aspects of the present invention, however, the object may be displayed in a manner that is truly independent from the operating system. (Specification – page 24, line 8 to page 29, line 10, and Abstract).

In some embodiments, the system of software components may be provided for graphically displaying an object in a manner independent from the operating system. The system of software components may be stored, for example, in a computer-readable storage device. In one example, the system of software components may include a graphics resource component (e.g., JButton 48, Fig. 8), a proxy component (e.g., JButton Proxy 88) and one or more peer components (e.g., JButton Peer 82 and/or JComponent Peer 84). The peer component may be adapted for receiving events pertaining to the object, and for routing the events to the proxy component. The proxy component may associate the object with the graphics resource component and invoke methods of the graphics resource component for displaying the object in a manner independent from the operating system. Because the graphics resource component does not use native code (i.e., OS-dependent code) for rendering images of the object upon a display, the graphics resource component may be considered a “lightweight software component.”

In some cases, the presently claimed peer and proxy components may also be independent from the operating system. For example, the presently claimed peer component may be independent from the operating system, but may be adapted to emulate the behavior of a second peer component, which does employ the windowing system of the operating system. In other words, the presently claimed peer component may be a “lightweight software component,” which performs a function similar to the heavyweight version provided by the second peer component. Unlike the presently claimed peer component, however, the second peer component may contain native code (i.e., OS-dependent code), which may be used to invoke the graphical resources of the operating system for creating an initial image of the object upon a display.

In some cases, the object may be part of a graphical user interface (GUI). When the object is included within a particular layout (e.g., a frame, scroll pane, or other container in the GUI), the association of the object and the graphics resource component may enable a parent-child relationship to be established between the layout and the lightweight graphics resource component. By establishing such a parent-child relationship, the presently claimed proxy component may allow the lightweight graphics resource component to draw an image of the object over the initial image of the heavyweight object, which was drawn with the aid of the windowing system of the operating system and the second peer component. By redirecting key events associated with the object (such as, e.g., focus, mouse, sizing, etc.) to the proxy component, the presently claimed peer component may enable the proxy component to respond to the key events, as if it were part of the layout containing the original heavyweight object. As a result, the presently claimed peer and proxy components may be used for displaying lightweight

versions of the heavyweight objects originally included with the application program, while maintaining object functionality and without modifying the original application program.

VI. ISSUES

1. Whether claims 1-17 are unpatentable under 35 U.S.C. § 103(a) over a web publication provided by Sun, entitled *Introducing Swing* (hereinafter “IS-SUN”) in view of a web publication, entitled *Mixing Heavy and Light Components*, by Amy Fowler (hereinafter “M-SUN”).

VII. GROUPING OF CLAIMS

Claims 1-6, 9-14 and 17 (Group I) stand or fall together.

Claims 7 and 8 (Group II) stand or fall together.

Claim 15 and 16 (Group III) stand or fall together.

The reasons why the three groups of claims are believed to be separately patentable are explained below in the appropriate parts of the Argument.

VIII. ARGUMENT

The presently claimed case provides a unique system of software components that enables an application program (e.g., a Java application) to be truly portable across all operating system (OS) platforms by providing various means for maintaining the “look and feel” of the application program. “A software program is said to be ‘portable’ across various platforms if the program can run without modification on any of those platforms.” (Specification, page 6, lines 23-24). Before disclosing such means, the specification highlights several drawbacks of prior art attempts at application portability.

For example, conventional Java application programs utilize a platform-dependent application program interface (API), commonly known as the Abstract Windowing Toolkit (AWT), to produce heavyweight software components that are written in native code (i.e., instructions specific to a particular OS). When AWT heavyweight software components are used for displaying images, the “look and feel” of those images may differ depending on the particular OS running the Java application program. In the context of a graphical user interface (GUI), “the ‘look and feel’ of a GUI refers to such things as the appearance, color and behavior of Buttons, TextFields, Listboxes, menus, etc...” (Specification, page 7,

line 4-8). Thus, there are certain limitations within the AWT that prohibit true portability of Java application programs, especially in terms of the look and feel of the application program. *See, e.g., Specification, page 7, line 24 - page 9, line 10; and page 18, line 10 - page 19, line 23.*

In an effort to overcome the platform-dependency of AWT-based programs, Swing was developed as part of the Java Foundation Classes. An API written using Swing contains no native code, and therefore, can be run on substantially any OS without changing the look and feel of the application. *See, e.g., Specification, page 9, lines 14-30, and page 19, line 25 to page 20, line 4.* Unfortunately, the lightweight software components of Swing cannot completely eliminate the platform-dependency of Java applications that use AWT. Since Swing versions of many AWT components (e.g., containers, such as frames) are unavailable, many programmers have attempted to mix Swing and AWT components within a given API. *See, e.g., Specification, page 21, lines 1-13.* However, straightforward mixing of Swing and AWT components tends to create many problems that programmers have simply learned to accept. *See, e.g., Specification, page 22, lines 1-28.*

One problem encountered when mixing Swing and AWT components within a given API arises when programmers attempt to add Swing components to an existing layout by allowing them to “draw” or “paint” over the platform-specific AWT renderings of objects in the layout. For example, a programmer may attempt to paint a lightweight button over a heavyweight button. However, the heavyweight button will still be visible, and may even obscure the lightweight button, since lightweight Swing components cannot successfully “compete” with pre-existing heavyweight components. Therefore, this approach fails to achieve the platform-independent “look” provided by Swing. *See, e.g., Specification, page 22, lines 13-21.*

Another problem encountered when mixing Swing and AWT components within a given API arises when programmers attempt to directly replace heavyweight AWT peer components with lightweight Swing counterparts which, instead of using the graphics resources of the operating system, utilize lightweight Swing objects for displaying platform-independent images of an object (contained, e.g., within a Frame). However, the lightweight Swing objects cannot respond to associated events (such as mouse events), because the lightweight Swing objects are not declared by the legacy application to be contained with the Frame. Therefore, this approach fails to achieve the platform-independent “feel” provided by Swing. *See, e.g., Specification, page 22, lines 23-28.*

Therefore, a need exists for an improved system and method that allows Swing and AWT components to be mixed within a given API, while preserving the “look and feel” of the platform-independent Swing components. *See, e.g., Specification, page 24, lines 8-21.*

The invention as recited in claims 1-17 overcomes the various difficulties connected with migrating legacy applications from AWT to Swing by providing a system of software components, referred to as “AWTSwing” components. In a general aspect, the presently claimed system of software components enable objects, which may be initially created by an AWT-based application program, to be displayed independently from the operating system running the AWT-based application.

As shown in Fig. 8, for example, the presently claimed system of software components may include a graphics resource component (e.g., JButton 48, Fig. 8), a proxy component (e.g., JButton Proxy 88) and one or more peer components (e.g., JButton Peer 82 and/or JComponent Peer 84). The proxy component associates the object with the graphics resource component. The peer component receives events (or method calls) pertaining to the object and routes the events to the proxy component. Upon receiving the events from the peer component, the proxy component functions to invoke the methods of the graphics resource component for displaying the object in a manner independent from the operating system. By redirecting method calls intended for the original heavyweight control (e.g., Button 30, Fig. 8) to the proxy component, an environment is created which allows the proxy component to function as if it were part of the layout (e.g., Frame 41, Fig. 8) containing the original heavyweight control. This approach enables Swing components to be successfully incorporated within an AWT-based application program, without modification of the application program. Thus, the presently claimed system of software components may enable the application program to be truly portable across all operating system platforms. *See, e.g., Specification, page 25, line 1 - page 28, line 16.*

Figs. 7 and 10 illustrate various differences between conventional methods (Fig. 7) and the presently claimed method (Fig. 10) for graphically displaying objects in a graphical user interface (GUI). Functional block diagrams are included within Figs. 7 and 10 to represent the flow of events and method calls that ensue when a user presses the “Ok” Button 31 included within Frame 41.

In Fig. 7, Frame 41 and all of its children are native objects of operating system (OS) 40. Therefore, pushing or activating “Ok” Button 31 with mouse cursor 43 causes an associated event to be detected by OS 40. In other words, the depression of “Ok” Button 31 causes a method call to be directed,

via heavyweight ButtonPeer 34, to an AWT library file (i.e., awt.dll 38), which determines (from the location of mouse cursor 43) if the region of the screen devoted to “Ok” Button 31 was selected when the event occurred. If so, event notification is routed from OS 40 to heavyweight ButtonPeer 34, which then invokes the methods of a native object (i.e., Button object 30) to change the state of “Ok” Button 31. By invoking the methods of a native object, heavyweight ButtonPeer 34 utilizes the graphical resources of the operating system (OS 40) to graphically display the “Ok” Button in a manner, which is clearly dependent on the operating system. *See*, e.g., Specification, page 23, line 1 to page 24, line 6.

In contrast, the presently claimed method of Fig. 10 may be used to graphically display an object in a manner, which is truly independent from the operating system. As shown in Figs. 8 and 10, for example, “Ok” Button 31 may be displayed by utilizing a lightweight graphics resource component (e.g., JButton 48, Fig. 8), instead of the native Button object (e.g., Button 30, Fig. 10) originally included within the legacy application program. In order to utilize the lightweight graphics resource component, the presently claimed method creates a lightweight proxy component (e.g., JButtonProxy 88, Fig. 10), which may be used to establish an association between the object and the lightweight graphics resource component. When events pertaining to “Ok” Button 31 (such as mouse events) are received by the presently claimed peer component (e.g., JButtonPeer 82, Fig. 10), they are routed to the proxy component. Upon receiving the events, the proxy component invokes the methods of the lightweight graphics resource component for displaying the “Ok” Button. By utilizing the lightweight graphics resource component, instead of the graphical resources of the operation system, the presently claimed method improves application portability by displaying objects in a truly independent manner. *See*, e.g., Specification, page 28, line 18 to page 29, line 10.

As described in more detail below, none of the cited art, either separately or in combination, provides motivation to teach or suggest the presently claimed system of software components, method or computer-readable storage device. Therefore, the teachings of the cited art cannot be used to render the limitations of the presently claimed case unpatentable.

ISSUE 1 ARGUMENTS

Patentability of Group I Claims 1-6, 9-14 and 17:

1. **IS-SUN and M-SUN each fail to provide teaching or suggestion for a system, computer-readable storage device, or method for graphical display of an object, where the system comprises a peer component, which is adapted to receive events pertaining to the object and route the events to a proxy component, which associates the object with a graphics resource component and invokes methods thereof to display the object in a manner independent from an operating system.**

Independent claim 1 states:

A system for graphical display of an object created by an application program running under an operating system, comprising: a graphics resource component adapted to display the object independently of the operating system; a proxy component, which associates the object with the graphics resource component and invokes methods of the graphics resource component to display the object; and a peer component, adapted to receive events pertaining to the object and route the events to the proxy component.

Independent claims 9 (i.e., a method for graphical display of an object) and 17 (i.e., a computer-readable storage device) recite similar limitations.

The article entitled *Introducing Swing* does just as the title implies – it briefly outlines the Swing architecture and the basic differences between Swing and AWT. However, *Introducing Swing* (otherwise referred to as IS-SUN) does not provide teaching or suggestion for a system, computer-readable storage device, or method for graphical display of an object, where the system comprises a peer component, which is adapted to receive events pertaining to the object and route the events to a proxy component, which associates the object with a graphic resource component and invokes methods thereof to display the object in a manner independent from an operating system, as taught in present claims 1, 9 and 17.

Statements in the Office Action mailed November 21, 2003 and in the final Office Action mailed May 27, 2004 admit that IS-SUN fails to provide teaching or suggestion for the presently claimed peer and proxy components. For example, the Examiner clearly states, “IS-SUN ... doesn’t teach a proxy component which associates the object with the graphics resource component and invokes methods of the graphics resource component to display the object, or a peer component, adapted to receive events pertaining to the object and route the events to the proxy.” (See, Office Action, page 2, Final Office

Action, pages 2-3). Appellants sincerely appreciate the Examiner's recognition of the lack of teaching within IS-SUN for the peer and proxy components recited in present claims 1, 9 and 17.

Though the Examiner admits to the lack of teaching within IS-SUN for the presently claimed peer and proxy components, the Examiner suggests that such teaching can be found within the article entitled *Mixing heavy and light components* (otherwise referred to as M-SUN). For example, the Examiner states, "M-SUN teaches a method of implementing swing components similar to that of IS-SUN, but further teaches a proxy component that associates an object with a graphics resource component, and further displays the object, in that the proxy component is the swing class (see page 2, paragraph 2), and a peer component, adapted to receive events pertaining to the object and route the events to the proxy component, in that the peer component is the ancestor (see page 2, paragraph 2)." (See, Office Action, page 2; final Office Action, page 3). In addition, the Examiner suggests that "it would have been obvious to one of ordinary skill in the art... to modify the swing component interface of IS-SUN to include the combinational properties as did M-SUN." (Office Action, page 3, Final Office Action, page 3). The Appellants respectfully disagree for at least the reasons set forth in more detail below.

M-SUN cannot be combined with IS-SUN to overcome the deficiencies therein, because M-SUN also fails to provide teaching or suggestion for the presently claimed peer and proxy components. Instead, M-SUN discloses various problems that may arise when programmers attempt to mix heavyweight AWT components with lightweight Swing components. Some of the problems disclosed by M-SUN are similar to those disclosed in the present specification. For example, M-SUN mentions some of the difficulties encountered when one attempts to add lightweight Swing components to an AWT layout by "painting" or "drawing" images of the Swing components over pre-existing AWT images. On page 3, M-SUN states that "[w]hen a lightweight component overlaps a heavyweight component, the heavyweight component is always on top, regardless of the relative z-order of the two components." On page 4, M-SUN illustrates how images rendered by heavyweight AWT components will obscure any overlapping images rendered by lightweight Swing components, and explicitly states that a "Swing label will never appear to be on top of [an] AWT label because the Swing label is rendering its contents in the native window of the frame (its first heavyweight ancestor), while the AWT label is rendering its contents in its own native window, which is actually a child of the frame's native window." As such, M-SUN recognizes a fundamental limitation of Swing, which does not permit images of Swing components to be drawn over AWT renderings.

The presently claimed case provides a solution to the above-mentioned problem by creating a proxy component, which is not included within the existing classes of software components belonging to the Swing application program interface (API). The presently claimed proxy component enables Swing components to be successfully incorporated within AWT-based application programs by associating the existing heavyweight (AWT) objects with lightweight (Swing) graphics resource components. This enables the proxy component to invoke the methods of the lightweight graphics resource components for displaying objects independently from the operating system. The presently claimed peer component differs from typical Swing peer components by routing events intended for the existing heavyweight object to the proxy component. *See, e.g., Specification, page 24, line 8 - page 29, line 10.*

Unlike the presently claimed case, M-SUN provides no solution to the problem other than sheer avoidance. For example, M-SUN states, “[b]ecause there’s sometimes no alternative to mixing heavyweight and lightweight components, we have provided a few options in Swing to make a certain level of component-mixing possible.” (M-SUN, page 3, paragraph 2). These options, or guidelines, include: “[d]o not mix lightweight (Swing) and heavyweight (AWT) components within a container where the lightweight component is expected to overlap the heavyweight one.” (M-SUN, page 5). From the above guideline, it is obvious that M-SUN chooses to avoid problematic situations, rather than present solutions to the problem. Since M-SUN fails to provide any solution of his own, M-SUN cannot be relied upon to disclose the presently claimed solution. In other words, M-SUN simply fails to provide any teaching or suggestion for the peer and proxy components recited in claims 1, 9, and 17.

In light of the Examiner’s suggestion that “the proxy component [of M-SUN] is the swing class” and “the peer component [of M-SUN] is the ancestor,” the Appellants wish to clarify the terms “class” and “ancestor” as they are used in the context of object-oriented programming and the presently claimed case. As described on pages 4 and 5 of the specification, a “class” is a fundamental entity in an object-oriented programming language that possesses certain attributes (e.g., shape, color, size, location on screen, behavior, etc.), which are common to all “objects” belonging to the class. The objects can be developed as modular building blocks, with subsequent objects referred to as “children” of higher-level “parent” objects. In some cases, a parent object may be a “class” of child objects, and the child objects may inherit the class attributes (i.e., the properties and methods) of the parent object, or “ancestor”, from which they derive. In the embodiment of Fig. 8, for example, Component 32 may be considered a “class” of objects, where one of the objects in the class is Button 30. Component 32 may also be considered to

be a “parent” or “ancestor” of Button 30, since various methods and/or properties of Button 30 may be derived from Component 32.

However, a “Swing class” cannot be considered to be a “proxy component” as suggested by the Examiner, since a “Swing class” merely defines the properties and methods (e.g., shape, color, size, location on screen, behavior, etc.) of a collection of Swing objects, whereas a “proxy component” actually functions to associate an object (e.g., a button displayed in a GUI) with a graphics resource component (e.g., JButton 48) and to invoke the methods of the graphics resource component (e.g., run the program code contained within JButton 48) for displaying the object. A “Swing class,” in itself, does not and cannot function to associate an object with a graphics resource component, invoke the methods of the graphics resource component, or display the object. Therefore, the mere mention of Swing classes within M-SUN does not provide teaching or suggestion for the proxy component recited in present claims 1, 9 and 17.

Furthermore, though a “peer component” may be an “ancestor” of some other object, merely stating so provides no evidence of the peer component being adapted to receive events pertaining to the object and to route the events to a proxy component. Since M-SUN fails to disclose a “proxy component”, any peer components that may be described by M-SUN cannot be configured to route events to a non-existent proxy component. As such, M-SUN also fails to provide teaching or suggestion for the peer component recited in present claims 1, 9, and 17.

The above arguments were presented in a Response to the Office Action mailed November 21, 2003. After reviewing the above arguments, the Examiner maintained his position in the final Office Action by providing yet another interpretation of the term “proxy component.” For example, statements in the final Office Action suggest that “the swing class [of M-SUN] can be considered a proxy component in the sense that a proxy can mean an element that acts as a substitute for another element (swing for AWT).” (Final Office Action, page 7). As described in more detail below, the Appellants disagree with the Examiner’s latest interpretation of a “proxy component,” and further assert that a “Swing class” cannot be considered a “proxy component,” as it is defined by the presently claimed case.

The Appellants recognize that, “[d]uring examination, the claims must be interpreted as broadly as their terms reasonably allow.” (See, MPEP 2111.01). However, the Appellants assert that the pending claims cannot be interpreted more broadly than their terms reasonably allow, or be given a

meaning that is inconsistent with the specification. The specification and present claims 1, 9 and 17 each describe the presently claimed “proxy component” as one that associates an object with a graphics resource component and invokes methods of the graphics resource component to display the object. Even when given its plain meaning, it is simply unreasonable for one skilled in the art to interpret the term “proxy component” as “an element that acts as a substitute for another element” since neither the specification nor the claims support such an interpretation. As such, the Examiner’s interpretation of the term “proxy component” is considered to be improper and erroneous.

In addition, the mere mention of Swing classes within M-SUN cannot be relied upon to provide teaching or suggestion for a proxy component, regardless of the manner in which the term “proxy component” is interpreted. As noted above, a “Swing class” merely defines the common properties and methods (e.g., shape, color, size, location on screen, behavior, etc.) associated with a collection of Swing objects that descend from the Swing class. However, a “Swing class” does not function to associate an object with a graphics resource component or invoke methods of the graphics resource component for displaying the object. Therefore, the mere mention of Swing classes with M-SUN does not provide evidentiary support for the presently claimed “proxy component”, as it is defined by the specification and the present claims. Furthermore, and contrary to the Examiner’s suggestions in the final Office Action, Swing classes are not “elements that act as substitutes for other elements,” and more specifically, do not substitute Swing components for AWT components. Therefore, a “Swing class” cannot be considered equivalent to a “proxy component,” even as it is improperly defined by the Examiner.

Further statements in the final Office Action appear to suggest that the presently claimed proxy component may be inherently taught by M-SUN. For example, the Examiner suggests that since M-SUN “teaches a combination of two APIs ... in order to combine two APIs there must be a connecting element.” (Final Office Action, page 7). Again, Appellants assert that it is improper to oversimplify and loosely interpret the presently claimed proxy component as a “connecting element,” since such interpretation is significantly more broad than allowed by the language used in claims 1, 9, and 17. Therefore, Appellants maintain the assertion that absolutely no teaching or suggestion for the presently claimed proxy component can be found, either explicitly or inherently described, within the sole reference (M-SUN) which the Examiner relies upon to provide such teaching.

2. There is no motivation to combine or modify the teachings of IS-SUN and M-SUN to provide the presently claimed peer and proxy components.

As explained above, IS-SUN and M-SUN each fail to provide teaching or suggestion for the presently claimed peer and proxy components, and therefore, cannot be combined to do so. In other words, even if the cited art were combined, the combined cited art would still fail to provide teaching or suggestion for the peer and proxy components described in present claims 1, 9, and 17.

In addition, the cited art cannot be modified to teach or suggest the aforementioned limitation, since IS-SUN and M-SUN each fail to suggest a desirability for doing so. The mere fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination [or modification]. *In re Mills*, 916 F.2d 680, 16 USPQ2d 1430 (Fed. Cir. 1990); MPEP 2143.01.

Unlike the presently claimed case, IS-SUN fails to mention the difficulties involved in mixing heavyweight AWT components and lightweight Swing components. Consequently, IS-SUN provides no desirability for overcoming such difficulties by providing lightweight peer and proxy components, as described in claims 1, 9 and 17. Though M-SUN does admit to some difficulties in mixing heavyweight AWT components and lightweight Swing components, M-SUN chooses to deal with the difficulties by avoiding the problematic situations altogether. By choosing avoidance, M-SUN fails to provide motivation for a system or method that actually solves the problem. Since IS-SUN and M-SUN each fail to provide motivation to teach or suggest the aforementioned limitations of the presently claimed case, IS-SUN and M-SUN cannot be modified to do so.

Obviousness can only be established by combining or modifying the teachings of the prior art to produce the claimed invention where there is some teaching, suggestion, or motivation to do so. *In re Fine*, 837 F.2d 1071, 5 USPQ2d 1596 (Fed.Cir. 1988); *In re Jones*, 958 F.2d 347, 21 USPQ2d 1941 (Fed. Cir. 1992); MPEP 2143.01. Appellants assert that the teaching or suggestion to make the claimed combination [or modification] must be found in the prior art, and not based on Applicant's own disclosure. *In re Vaeck*, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991); MPEP 2142.

3. The Examiner has failed to adequately support and/or establish a *prima facie* ground of obviousness.

To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all claim limitations. MPEP § 2143. None of these three criteria have been met by the Examiner in the present case. First of all, no suggestion or motivation to modify the cited references can be found within the cited references to teach or suggest the aforementioned limitations of present claims 1, 9 and 17, as explained above in Argument 2. The criterion of a reasonable expectation of success cannot be met if no teaching, suggestion or motivation exists, because there is then nothing at which to be successful. Finally, none of the cited art, either alone or in combination, teaches all of the limitations of claim 1, 9 and 17, as explained above in Argument 1. The third criterion recited above has therefore also not been met, and a *prima facie* case of obviousness has not been established.

Conclusion

As explained in Arguments 1-3 above, at least some limitations of claims 1, 9, and 17, as well as at least some limitations of claims 2-8 and 10-16, are not taught or suggested by the cited art. Furthermore, there is no teaching, suggestion or motivation to modify the cited art to teach the limitations of these claims. For at least the reasons set forth above, claims 1-17 are patentably distinct over the cited art, and the rejection of Group I claims 1-6, 9-14 and 17 under 35 U.S.C. § 103(a) is asserted to be erroneous.

Patentability of Groups II and III Claims 8 and 16:

Because claims 7 and 8 of Group II and claims 15 and 16 of Group III are respectively dependent from claims 1 and 9 of Group I, the arguments presented above for patentability of claim 1 apply equally to claims 7, 8, 15, and 16, and are herein incorporated by reference. In addition to the arguments presented above with respect to claim 1, arguments are provided below to further establish the patentability of the current claims under 35 U.S.C. § 103(a).

2. **None of the cited art teaches or suggests that, when the object is part of a layout, the association of the object with the graphics resource component can be used to establish a parent-child relationship between the layout and the graphics resource component, which can then be used to allow the lightweight graphics resource component to draw over an existing image of the object drawn with the aid of the windowing system of the operating system.**

Claim 7 recites, “wherein the object is part of a layout, and the association of the object with the graphics resource component establishes a parent-child relationship between the layout and the graphics resource component.” Claim 8 recites, “wherein the parent-child relationship between the layout containing the object and the graphics resource component allows the graphics resource component to draw over an existing image of the object drawn with the aid of the windowing system of the operating system.” Claims 15 and 16 recite similar limitations. As such, claims 7 and 8, as well as 15 and 16 place additional limitations on the proxy component introduced in claims 1 and 9, respectively, by describing how the proxy components may establish a parent-child relationship, which allows lightweight (i.e., Swing) graphics resource components to “draw” or “paint” over existing images of heavyweight (i.e., AWT) objects drawn with the graphical resources of the operating system.

As noted above in the arguments of Group I claims, IS-SUN and M-SUN each fail to disclose the presently claimed peer and proxy components, and furthermore, cannot be modified or combined to do so. As described in more detail below, IS-SUN and M-SUN also fail to provide teaching or suggestion for the additional limitations placed on the presently claimed proxy component, as set forth in dependent claims 7, 8, 15, and 16.

Statements in the final Office Action rely on M-SUN for teaching the limitations of dependent claims 7, 8, 15, and 16. For example, the final Office Action states, “M-SUN teaches, in page 4, paragraph 2 and page 6, paragraph 2 and the following picture, that if heavyweight components are used it is possible for them to obscure what is drawn by the windowing system of the operating system.” (Final Office Action, page 5). The Appellant agrees with the Examiner’s interpretation of M-SUN. However, the Examiner’s interpretation is completely inconsistent with the limitations of dependent claims 7, 8, 15, and 16, which describe how a parent-child relationship may allow a lightweight graphics resource component (*see*, claims 1 and 9) to draw over an existing image of a (heavyweight) object drawn with the aid of the operating system. Because M-SUN clearly states that lightweight Swing objects “will never appear to be on top” of heavyweight AWT objects (*see*, pages 3-4 of M-SUN), M-SUN cannot be relied

upon to provide teaching or suggestion for the additional limitations set forth in dependent claims 7, 8, 15, and 16.

2. There is no motivation to modify or combine the cited art to teach or suggest the presently claimed parent-child relationship.

As noted above, none of the cited art provides teaching or suggestion for the presently claimed parent-child relationship, which allows a (lightweight) graphics resource component to draw over an existing image of a (heavyweight) object drawn with the aid of the operating system.

In addition to the lack of teaching, Appellants assert that the cited art cannot be combined or modified to teach or suggest the presently claimed parent-child relationship, since the cited art fails to suggest a desirability for doing so. The mere fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination. *In re Mills*, 916 F.2d 680, 16 USPQ2d 1430 (Fed. Cir. 1990); MPEP 2143.01.

As noted above, M-SUN fails to provide a manner in which lightweight graphics resource components can be used to successfully draw over images previously drawn with the aid of the operating system. Instead, M-SUN strongly recommends that programmers “[d]o not mix lightweight (Swing) and heavyweight (AWT) components within a container where the lightweight component is expected to overlay the heavyweight one.” (M-SUN, page 4, paragraph 3). By choosing avoidance, rather than presenting a solution to the problem, M-SUN provides no desirability for the presently claimed parent-child relationship. As such, M-SUN cannot be modified to teach or suggest the additional limitations set forth in dependent claims 7, 8, 15, and 16.

3. The Examiner has failed to adequately support and/or establish a *prima facie* ground of obviousness.

To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all claim limitations. MPEP § 2143. None of these three criteria have been met by the Examiner in the present case. First of all, no suggestion or motivation to modify the cited references can be found within the cited references to teach or suggest the aforementioned

limitations of claims 7, 8, 15, and 16 as explained above in Argument 2. The criterion of a reasonable expectation of success cannot be met if no teaching, suggestion or motivation exists, because there is then nothing at which to be successful. Finally, none of the cited art, either alone or in combination, teaches all of the limitations of claims 7, 8, 15, and 16, as explained above in Argument 1. The third criterion recited above has therefore also not been met, and a *prima facie* case of obviousness has not been established.

Conclusion

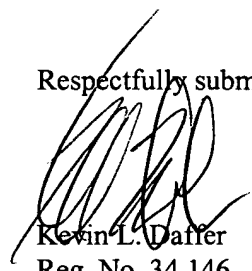
As explained in Arguments 1-3 above, at least some limitations of claims 7, 8, 15, and 16 are not taught or suggested by the cited art. Furthermore, there is no teaching, suggestion or motivation to modify the cited art to teach the limitations of these claims. For at least the reasons set forth above, claims 7, 8, 15, and 16 are patentably distinct over the cited art, and the rejection of Group II claims 7, 8, 15, and 16 under 35 U.S.C. § 103(a) is asserted to be erroneous.

IX. CONCLUSION

For the foregoing reasons, it is submitted that the Examiner's rejection of claims 1-17 was erroneous, and reversal of the Examiner's decision is respectfully requested.

The Commissioner is hereby authorized to charge the required fee(s) to Conley Rose deposit account 50-3268/5468-07300.

Respectfully submitted,



Kevin L. Daffer
Reg. No. 34,146
Attorney for Appellant

Conley Rose, P.C.
P.O. Box 684908
Austin, TX 78768-4908
Date: October 25, 2004

X. APPENDIX

The present claims on appeal are as follows.

1. A system for graphical display of an object created by an application program running under an operating system, comprising:

a graphics resource component adapted to display the object independently of the operating system;

a proxy component, which associates the object with the graphics resource component and invokes methods of the graphics resource component to display the object; and

a peer component, adapted to receive events pertaining to the object and route the events to the proxy component.

2. The system as recited in claim 1, wherein the peer component is independent of the operating system, and emulates the behavior of a second peer component that employs the windowing system of the operating system.

3. The system as recited in claim 1, wherein the object is part of a graphical user interface (GUI) associated with the application program.

4. The system as recited in claim 3, wherein a look and feel of the GUI is independent of the operating system.

5. The system as recited in claim 1, wherein the application program is written in Java programming language.

6. The system as recited in claim 1, wherein the proxy extends an existing class of software components belonging to the Swing application program interface (API).

7. The system as recited in claim 1, wherein the object is part of a layout, and the association of the object with the graphics resource component establishes a parent-child relationship between the layout and the graphics resource component.

8. The system as recited in claim 7, wherein the parent-child relationship between the layout containing the object and the graphics resource component allows the graphics resource component to draw over an existing image of the object drawn with the aid of the windowing system of the operating system.

9. A method for graphical display of an object created by an application program running under an operating system, comprising:

utilizing a graphics resource component adapted to display the object independently of the windowing system of the operating system;

creating a proxy component and establishing an association between the object and the graphics resource component via the proxy component;

receiving events pertaining to the object in a peer component and routing them to the proxy component; and

in response to the events, invoking methods of the graphics resource component via the proxy component to display the object.

10. The method as recited in claim 9, wherein the peer component is independent of the operating system, and the method further comprises emulating the behavior of a second peer component that employs the windowing system of the operating system.

11. The method as recited in claim 9, wherein the object is part of a graphical user interface (GUI) associated with the application program.

12. The method as recited in claim 11, wherein a look and feel of the GUI is independent of the operating system.

13. The method as recited in claim 9, wherein the application program is written in Java programming language.

14. The method as recited in claim 9, wherein the proxy extends an existing class of software components belonging to the Swing API.

15. The method as recited in claim 9, wherein the object is part of a layout, and the method further comprises using the association of the object with the graphics resource component to establish a parent-child relationship between the layout and the graphics resource component.

16. The method as recited in claim 15, further comprising using the parent-child relationship between the layout containing the object and the graphics resource component to allow the graphics resource component to draw over an existing image of the object drawn with the aid of the windowing system of the operating system.

17. A computer-readable storage device, comprising:

a windows-based operating system;

an application program running under the operating system;

a graphics resource component adapted to display an object created by the application program independently of the windowing system of the operating system, by:

creating a proxy component and establishing an association between the object and the graphics resource component via the proxy component;

receiving events pertaining to the object in a peer component and routing them to the proxy component; and

in response to the events, invoking methods of the graphics resource component via the proxy component to display the object.